Preventing Bug Reopening With Effective Software Verification

Rodrigo Souza Christina Chavez Roberto Bittencourt

July 4, 2013

Abstract

Many companies perform software verification on bug fixes to determine if they are appropriate. Verification can spot missed corner cases and other problems, thus preventing the shipping of buggy code. Software verification processes vary in dimensions such as time (when is verification performed?), people (is there a specific role responsible for verifications?), and technique (code review, ad hoc testing, and so on). What is the optimal software verification process so that it detects most problems with bug fixes before they are released to the public? Answering this question would be important to software companies, that would release higher quality products, as well as to the research community.

1 Project Summary

Fixing bugs is an important software development task, often supported by bug tracking systems [5]. In a typical scenario, a bug report is created, then the bug is fixed, and finally the report is closed. Sometimes, however, someone discovers that the fix was incomplete or innapropriate. In this case, the bug report is reopened so it can be further investigated.

Reopened bugs take twice as much time to fix, on average [6], and involves the participation of 20 to 60% more developers [4]. Even worse is when a bug receives an incomplete fix but developers only notice it after the software has been released. In this case, the bug is reopened only after the release, already too late, and the software quality, as perceived by users, is impaired.

To avoid post-release bug reopenings, in some projects the bug fixes are verified. Verifying a bug fix means that someone independently checks if it is appropriate and complete. If it is not, the bug report is reopened right away, so it can be properly fixed before the next release.

Of course, verification is not perfect, and a bug can be reopened even after it has been considered appropriate during its verification. The verification, in this case, can be considered unsuccessful.

Why some verifications are successful and others are not? Although many factors may contribute to the outcome, such as source code complexity [1] and developer experience [4], many of them cannot be easily controlled. Therefore, we should ask: how to improve the bug fix verification success rate?

1.1 Objectives

We propose to investigate what are the dimensions of software verifications processes and how to improve the success rate by tailoring the process along these dimensions. In a previous work [8], we revealed dimensions such as time (are bugs fixes verified at a constant rate or in bursts?), people (are verifications performed by specialized teams?), and technique (do verifications involve automated testing, code review or other techniques?).

There is little empirical evidence, however, on how these and other dimensions affect the success rate of verifications. In this project, we intend to expand on these dimensions and determine how they can help improve the software verification process.

1.2 Intellectual Merit

Recent works have tried to predict bug reopening using factors related to the bug, the bug fix, and the developers (e.g., [7, 12]). However, most findings relate bug reopening to factors that cannot be easily controlled. Therefore, the results cannot be used to prevent reopening.

In this project, we aim to determine how reopening can be made less frequent by improving the verification process. Although some studies try to empirically determine the relative effectiveness of distinct verification techniques, such as automated testing and code review [11], our approach differs in two aspects: methodology and scope.

Traditionally, empirical studies of verification techniques are based on experiments or quasi-experiments, and thus the results may fail to generalize or to apply to specific scenarios. In our project, we will leverage software development data, specifically bug reports, in order to find results that are relevant to a specific organization or development team.

Regarding scope, while most studies focus on a specific aspect of the software verification process—namely, the verification technique—, we will to adopt a more holistic approach. We plan to study the influence of different dimensions of the software verification process—e.g., time and people—, on its effectiveness.

1.3 Broader Impact

The expected tangible outcome of this project is a set of empirically supported guidelines to help improve the effectiveness of software verification processes. They have, therefore, the potential to improve both the productivity of software development and the quality of software products.

The productivity would be increased by reducing the number of bugs that would be reopened after a failed verification, thus reducing the amount of rework. Besides, the quality would be improved by detecting problems with bug fixes before the next release, during verification.

2 Background

The term reopening comes from bug tracking systems such as Bugzilla[1]. In those systems, bugs that are not resolved are considered open. Thus, if a bug is resolved and, later on, someone finds that the resolution is not appropriate, the bug is reopened.

Usually a developer resolves a bug by fixing it, but this is not the only possible resolution. During bug triage, a bug report can be resolved by tagging it as invalid (i.e., it is not a bug), incomplete (there is not enough information to reproduce it), "won't fix" (the bug is not worth fixing), among other resolutions.

Therefore, in the broad sense, a bug may be considered reopened if its triage was not definitive, e.g., if the report was initially incomplete, but new information was added afterwards. In this proposal, we use the term reopening in a stricter sense, meaning the bug was considered resolved by applying a fix, but the fix was later considered innapropriate.

3 Related Work

Despite its importance, only recently researchers have started to study bug reopening. Shihab and colleagues [7, 6] developed a decision tree model, based on features about the bug report, the bug fix, and human factors, to predict which bugs would be reopened (in the broad sense). They found that certain components are more prone to reopening (such as those dealing with concurrency).

Zimmermann and colleagues [12] asked Microsoft engineers what are common causes for bug reopening (in the broad sense). Responses included the difficulty to reproduce a bug, the misunderstanding of root causes, the lack of information in the initial report, the increase of the bug priority, incomplete fixes, and code integration problems.

They also have built classification models to predict reopened bugs. They found that bugs reported by users are more likely to be reopened than those discovered through code review or static analysis, supposedly because those reported by users are harder to and reproduce and tend to be more complex. According to the authors, other factors that favor reopening include bug severity and geographical distribution of developers participating in the bug.

Both Shihab's [7, 6] and Zimmermann's [12] works investigate reopening in a broad sense. In our opinion, the reopening of fixed bugs and the reopening of bugs that were just triaged are different phenomena, with distinct causes and prevention approaches. In this proposal, we aim to investigate the reopening of fixed bugs.

Some works deal specifically with bugs that were reopened after receiving a fix. Jongyindee and colleagues [3] found that bugs are less likely to be reopened when fixed by more active developers. Almossawi [1] concluded that bugs located in code with high cyclomatic complexity are more likely to be reopened after fix.

Most findings in literature are of limited utility for those trying to prevent bug reopening, because they relate reopening to uncontrollable factors. For example, one cannot directly avoid that users discover bugs [12] that reside in code with high cyclomatic complexity [1] or in certain components [7], nor it is feasible that only the most active developers fix those

[1] `http://www.bugzilla.org/`

bugs [3]. In this proposal, we are interested in ways of minimizing post-release reopening by improving a directly controllable aspect of software development: the software verification process.

4 Data and Methods

Based on Figure 3 of CODEMINE's whitepaper [2], the project relies on the following types of data: work item, organization, code review, process information, and, possibly, build and test (if they can be related to work items). The analysis will be centered on work items, and the other types of data will provide context (e.g., developer role at Microsoft, how bug was discovered [12]) to allow more refined inferences. The project would also benefit from interviews or surveys with Microsoft engineers.

We expect to conclude the project in 8 weeks, according to the following schedule: (1) learn about CODEMINE schema, API and tools; (2) identify verification practices at Microsoft through surveys and data analysis; (3) write and run scripts to clean up data; (4) write and run scripts to transform data; (5) write reports to characterize the data; (6) write and run analyses; (7) check findings with Microsoft engineers; (8) refine analyses.

5 People and Related Publications

One PhD student and two professors are involved in the project. Below, a short bio for each one.

Rodrigo Souza is a computer science PhD student at Federal University of Bahia, Brazil, where he received his bachelor's degree, and has a M.Sc. degree from the Federal University of Campina Grande, Brazil. He has already done some work to support this proposal as part of his thesis, using data from open source projects, and currently contributes to the Data Analysis Community Portal[2].

Christina Chavez is ...

Roberto Almeida Bittencourt received a degree in Electrical Engineering from the Federal University of Paraiba, Brazil, a M.Sc. degree in Computer and Systems Engineering from Linköping University, Sweden, and a Ph.D. degree in Computer Science from the Federal University of Campina Grande, Brazil. He is currently an assistant professor at the State University of Feira de Santana (UEFS). He has published 20 papers in refereed conferences and workshops. His main research interests are in software evolution, mining software repositories, software engineering education and collaborative systems.

We have published three papers that supports this proposal. The first one was a characterization of the verification process of open source projects using data from bug reports. The results were presented at the 9th Working Conference on Mining Software Repositories (MSR 2012) [8].

This year we presented two papers [9, 10] at the Data Analysis Patterns in Software Engineering (DAPSE) workshop, organized by Microsoft Research and held in San Francisco,

[2] `http://dapse.unbox.org/`

together with the 35th International Conference on Software Engineering (ICSE 2013). The papers include four patterns that help data scientists clean and transform bug data before it can be analyzed.

References

[1] A. Almossawi. Investigating the architectural drivers of defects in open-source software systems: an empirical study of defects and reopened defects in gnome, 2012.

[2] J. Czerwonka, N. Nagappan, W. Schulte, and B. Murphy. CODEMINE: Building a software analytics platform for collecting and analyzing engineering process data at Microsoft. Technical Report MSR-TR-2013-7, Microsoft Research, 2013.

[3] A. Jongyindee, M. Ohira, A. Ihara, and K.-i. Matsumoto. Good or bad committers? a case study of committers' cautiousness and the consequences on the bug fixing process in the eclipse project. In *Proceedings of the 2011 Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement*, IWSM-MENSURA '11, pages 116–125, Washington, DC, USA, 2011. IEEE Computer Society.

[4] J. Park, M. Kim, B. Ray, and D.-H. Bae. An empirical study of supplementary bug fixes. In M. Lanza, M. D. Penta, and T. Xi, editors, *MSR*, pages 40–49. IEEE, 2012.

[5] R. Patton. *Software Testing (2nd Edition)*. Sams, Indianapolis, IN, USA, 2005.

[6] E. Shihab. *An Exploration of Challenges Limiting Pragmatic Software Defect Prediction*. PhD thesis, School of Computing, Queen's University, Kingston, Ontario, Canada, 2012.

[7] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K.-i. Matsumoto. Predicting re-opened bugs: A case study on the eclipse project. In *Proceedings of the 2010 17th Working Conference on Reverse Engineering*, WCRE '10, pages 249–258, Washington, DC, USA, 2010. IEEE Computer Society.

[8] R. Souza and C. Chavez. Characterizing verification of bug fixes in two open source ides. In M. Lanza, M. D. Pent, and T. Xi, editors, *MSR*, pages 70–73. IEEE, 2012.

[9] R. Souza, C. Chavez, and R. Bittencourt. Patterns for cleaning up bug data. In *Proceedings of the First Workshop on Data Analysis Patterns in Software Engineering*. IEEE, May 2013.

[10] R. Souza, C. Chavez, and R. Bittencourt. Patterns for extracting high level information from bug reports. In *Proceedings of the First Workshop on Data Analysis Patterns in Software Engineering*. IEEE, May 2013.

[11] J. W. Wilkerson, J. F. Nunamaker, Jr., and R. Mercer. Comparing the defect reduction benefits of code inspection and test-driven development. *IEEE Trans. Softw. Eng.*, 38(3):547–560, May 2012.

[12] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy. Characterizing and predicting which bugs get reopened. In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 1074–1083, Piscataway, NJ, USA, 2012. IEEE Press.